UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Trabajo Final de Grado

Grado en Ingeniería Electrónica Industrial y Automática

APLICACIÓN MÓVIL DE SOPORTE A LA PREINSCRIPCIÓN UNIVERSITARIA (I)

Tutor: Pablo Fernández Duran
Cotutor: Jordi Marco Gómez

Autor: Laura Molina López

Convocatoria ordinaria 2019/ 20-2

Abstract

Diseño y programación de una aplicación móvil de soporte a la preinscripción universitaria. Dicha aplicación tiene el nombre de **Graus UPC**.
Esta aplicación consta de un diseño, de su programación tanto visual como funcional, y de una base de datos a tiempo real. Su principal objetivo es ayudar a los futuros universitarios a elegir qué carrera quieren cursar.

Sus funcionalidades básicas son mostrar el listado de los grados universitarios de toda la UPC, poder filtrar el listado, poder guardar los grados favoritos en un listado personal y mostrar un calendario con las fechas de los exámenes de las PAU.

Este proyecto tiene presente tanto los aspectos visuales de la aplicación como su eficiencia y rapidez, para así enfocarse en la experiencia global del usuario.
Graus UPC ha sido programada junto a Martí Ollé Alavedra.

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Este proyecto fue realizado por:

Laura Molina Lopez

Director:

Pablo Fernández Duran

Co-director:

Jordi Marco Gómez

TFG Trabajo Fin de Grado

Fecha de entrega: 30 Junio 2020

UPC Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa (**ESEIAAT**)

Agradecimientos

En primer lugar debo agradecer a mi director de proyecto Pau Fernández, por su excelente trabajo de guía, consejo y motivación para realizar este TFG. También dar las gracias a mi compañero Martí Ollé por su buen carácter y ganas de trabajar, es un crack.

Gracias también a mi madre y mi padre, por su constante apoyo y ánimos a nunca rendirme.

Finalmente agradecer a todos mis amigos y a las personas que han hecho esto posible, tanto este trabajo como la realización de mi carrera.

¡Muchas gracias!

Laura.


UPC
GRAUS
UPC

Índice de contenidos

Índice de figuras

1. Introducción

a. Objeto

Diseño e implementación de una aplicación móvil que permite visualizar el listado de todos los grados universitarios de la universidad politécnica de Catalunya (UPC). Esta aplicación se llama "Graus UPC" y ha sido realizada junto a Martí Ollé Alavedra.

Con esta aplicación móvil se quiere ayudar a los futuros universitarios a elegir qué carrera quieren cursar.

La aplicación ha sido realizada con Flutter y está conectada a una base de datos a Firebase. La app tiene un trabajo de diseño detrás, ya que, tiene mucho en cuenta la UX (user experience o experiencia de usuario). En esta memoria está documentado que es Flutter y Firebase, y la aplicación de UX en el diseño de la app.

Este proyecto se puede resumir en tres apartados básicos:

- Diseño de la aplicación y sus funcionalidades
- Aplicación móvil realizada con Flutter
- Base de datos de Firebase

b. Alcance del proyecto

La app debe ser útil para cualquier persona que quiere cursar un grado universitario dentro del territorio de Cataluña. Se pretende facilitar a los estudiantes la búsqueda de información de los grados universitarios para mejorar el proceso de elección de sus estudios.

Aunque su principal objetivo no sea cursar un grado UPC, quizás con la app puede encontrar información interesante y útil.

c. Justificación del proyecto

Durante los años que cursé el bachillerato no tenía ni idea de qué carrera quería hacer o a que me quería dedicar. Cada semana quería hacer un grado diferente y me pasaba horas en diferentes webs, como por ejemplo educaweb, mirando que podría estudiar.

Cada vez es más frecuente escuchar alumnos universitarios indecisos, que dudan si han escogido correctamente sus estudios, y también muchos universitarios que cambian de carrera más de una vez. Y finalmente, gente que ha cursado un módulo y se plantean ampliar sus estudios con un grado universitario.

Toda esta gente sólo puede buscar la oferta que tienen de grados por sitios webs, a menudo muy poco intuitivos. Cuando encuentran un grado que les interesa, hay que abrir otra pestaña del buscador para visualizar la web oficial del grado de la universidad, y así hasta tener muchas pestañas abiertas en el navegador(a mí me pasaba muchísimo, ya que estaba muy indecisa) . Y cuando encuentras un grado que te gusta tienes que hacer una captura de pantalla o crear un documento donde hacer tu listado manualmente e introducir tu mismo la información del grado.

Con una app con el listado de grados universitarios todo esto sería mucho más sencillo, facilitando también el acceso a los listados, ya que se podría hacer con un dispositivo móvil de una forma mucho más cómoda y accesible.

Es una herramienta que no existe en el mercado y que podría tener una alta demanda para su uso. Por este motive, quise crear Graus UPC.

d. Requisitos básicos

Las funcionalidades básicas de la app son:

- Mostrar el listado de los grados universitarios de toda la UPC.
- Poder filtrar el listado por diferentes campos:
    - Nota de corte
    - Localización
    - Rama (Tecnológico, científico, social, artístico, humanístico)
    - Modalidad (Presencial, semipresencial o no presencial)
- Filtrar el listado por palabras y/o letras.
- Poder guardar los grados favoritos en un listado personal.
- Mostrar un calendario con las fechas de los exámenes de las PAU. (Matrículas, pruebas, revisiones de examen, etc.)

Otras funcionalidades:

- Poder compartir con otros compañeros el grado que te ha gustado, a través de diferentes redes sociales como WhatsApp o Telegram.
- Tener un perfil de usuario donde tener tu foto e información.
- Poder registrarse en la app con tu cuenta de Google.

2. Planificación

Este proyecto ha sido realizado por dos personas, por este motivo la planificación ha sido clave. Primero se pensó en usar un software parecido a Trello, que se usa para apuntar tareas y su estado. Finalmente, nos supimos entender muy bien y no hizo falta.

Las partes del proyecto se han repartido equitativamente:

	Martí	Laura
Diseño de la aplicación		
Introducción de datos a Firebase		
Home Screen, listado de grados		
FiltreScreen		
Búsqueda por nombre del grado		
InfoScreen, calendario		
Sign in with Google		
Listado de favoritos		
Profile Screen, pantalla usuario		
Ficha Screen, con la ficha del grado		
Filtros de la lista y sus respectivas pantallas		
Frontend* general		
Backend* general		

Los desarrolladores **frontend crean el aspecto de un sitio web.*
Los desarrolladores de **backend crean cómo funciona un sitio web*

Al principio se barajó la idea de repartir el proyecto por frontend y backend, pero al final los dos hemos trabajado tanto en la programación del diseño, de las funcionalidades y de la base de datos de la aplicación.

Se realizó un diagrama de Gantt para poder ver de forma global el proyecto y su estado, así como el tiempo dedicado en cada tarea:

/* Ver **ANNEX1_GANTTCHART.pdf** */

3. Estado del arte

a. Aplicaciones parecidas

Actualmente no hay ninguna aplicación parecida en el mercado. No hay ni la versión de todas las carreras disponibles en todo el estado.

Tan solo hay webs con la que puedes filtrar el listado de grados disponibles en España o Catalunya sin poder tener tu listado de favoritos guardados de forma permanente. Algunas de estas webs son oficiales del gobierno y otras de carácter privado:

- Canal universitats - Gencat (Catalunya)
  http://universitats.gencat.cat
- unportal (Catalunya)
  https://graus.unportal.net/wb/unportal/es/buscador/index.html
- Educaweb (España)
  https://www.educaweb.com/
- infoeducación (España)
  https://infoeducacion.es/carreras-universitarias-espana/
- notasdecorte.es (España)
  https://notasdecorte.es/
- yaq.es (España)
  https://yaq.es/carreras-universitarias

En Play Store (el mayor sitio de descargas de aplicaciones Android) no hay ninguna aplicación móvil parecida a ninguna a estas webs. Tampoco hay disponible ninguna app para ayudar a escoger carrera y tener el listado de los grados en tu dispositivo móvil.

En concreto Graus UPC se centra en los grados que ofrece la universidad politécnica de Catalunya, siendo la única universidad con una aplicación móvil donde se muestre todos sus estudios disponibles.

b. Comparaciones y estudio de estas

Todas las aplicaciones mínimamente parecidas a la app realizada en este proyecto son aplicaciones web. Muchas de estas aplicaciones web no están bien adaptadas para dispositivos móviles.

Solo la web de Canal Universitats de Gencat.cat tiene una buena adaptación a dispositivos móviles y tiene una opción para guardar tus grados favoritos como en una cesta de la compra que luego puedes mandarte a tu mail personal.

Esta funcionalidad ha sido añadida hace muy poco tiempo, después de empezar este proyecto, por este motivo seguramente la versión que hay actualmente no sea la definitiva.

Por ahora tiene este aspecto en un dispositivo Android:

Figura 1

Figura 2

Figura 1, se muestra el listado por orden alfabético al ser filtrado por la rama de estudio de Ingeniería y Arquitectura.

Figura 2, menú inicial de la aplicación web.

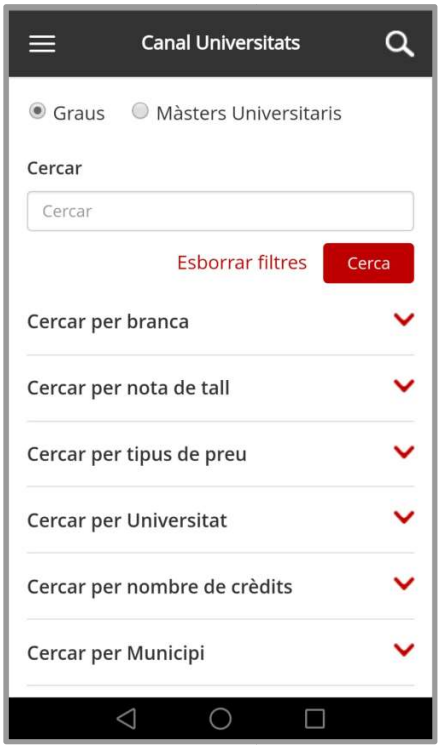

Figura 3

Figura 4

Figura 3, buscador de grados y filtros a seleccionar.

Figura 4, pantalla de estudios favoritos que puedes compartir por redes sociales.

Análisis general de la aplicación web:

Aspectos positivos	Aspectos negativos
• Aplicación web bien adaptada para dispositivos móviles. • Aplicación sencilla. • Diseño minimalista sin información de más. • Datos actualizados desde la misma fuente gencat.cat • Funcionalidades rápidas y buena carga de la información.	• Aplicación poco intuitiva. • Pocas funcionalidades. • Fallos en el buscador por palabra clave, no está muy bien calibrado y da algunos fallos de búsqueda. • Diseño básico y poco atractivo. • No muestra el listado de forma completa si no haces algún tipo de búsqueda.

Con este pequeño estudio de la aplicación se pueden sacar conclusiones para la realización de este proyecto:

1. Graus UPC debe tener parecidos con la web gencat ya que es la aplicación web que cumple más con los requisitos de este proyecto.

2. Se tiene que trabajar un diseño atractivo y visual para que la gente quiera usar la app. Se debe mostrar el listado completo de los grados, y los filtros deben funcionar a la perfección, esta funcionalidad será el centro de la aplicación móvil, y nunca se debe olvidar que ese es el mayor objetivo de la app.

3. Su estructura debe ser sencilla y clara, y su uso muy intuitivo. También se debe tener una base de datos muy clara y concisa, para disminuir el riesgo de errores. Las llamadas para cargar datos o actualizarlos deben ser mínimas para que la aplicación sea rápida y consuma poca RAM*.

*RAM: Sigla de Random Access Memory (‘memoria de acceso aleatorio’), memoria principal de la computadora, donde residen programas y datos, sobre la que se pueden efectuar operaciones de lectura y escritura.

4. Diseño

a. UX/UI

A pesar de lo que muchos piensan, UX y UI son conceptos diferentes. Sin embargo, están muy relacionados. Y la combinación de ambos es esencial para crear un diseño ideal para el consumidor.

Para entender mejor todo esto vamos a empezar por el principio.

¿Qué es UX?

UX (User Experience) hace referencia a la forma en la que los usuarios interactúan con un producto o servicio. Es decir, cómo y para qué un usuario utiliza un objeto o interactúa con una web o app.
Sin duda, la esencia del diseño UX está en el conocimiento de los usuarios. En otras palabras, para crear un buen diseño UX hay que comprender las necesidades de los usuarios y, por supuesto, satisfacerlas de una forma simple y clara. Por lo tanto, un buen resultado es aquel que es útil para el usuario.

¿Qué es UI?

Por otro lado, el diseño UI o User Interface se centra en la parte visual. Es decir, si UX se encarga de que un producto sea útil para los usuarios, UI lo hace atractivo y visual.

Los colores, la tipografía, las imágenes son algunos de los elementos con los que trabaja el diseñador UI para hacer que un producto sea atractivo. Pero de nada sirve tener un producto bonito si no satisface las necesidades de los usuarios para los que está pensado. Por eso, UX y UI deben ir de la mano para lograr un producto 100% pensado para los clientes.

Veámoslo con un ejemplo. Piensa en la bandeja de entrada de tu correo electrónico. Todo lo que ves es el trabajo de un diseñador de UI. Ahora imagina que intentas abrir un correo pero la página tarda mucho o que no puedes encontrar ese contacto tan importante al que tienes que escribir. Pues los encargados de que no te desesperes en los procesos son los diseñadores UX.

En este ejemplo, tanto UX como UI son componentes críticos para que el usuario disfrute del producto. Por este motivo el trabajo de un UX/UI Designer es tan importante. De su trabajo depende, en parte, la imagen que el usuario va a tener de la empresa.

UX vs. UI

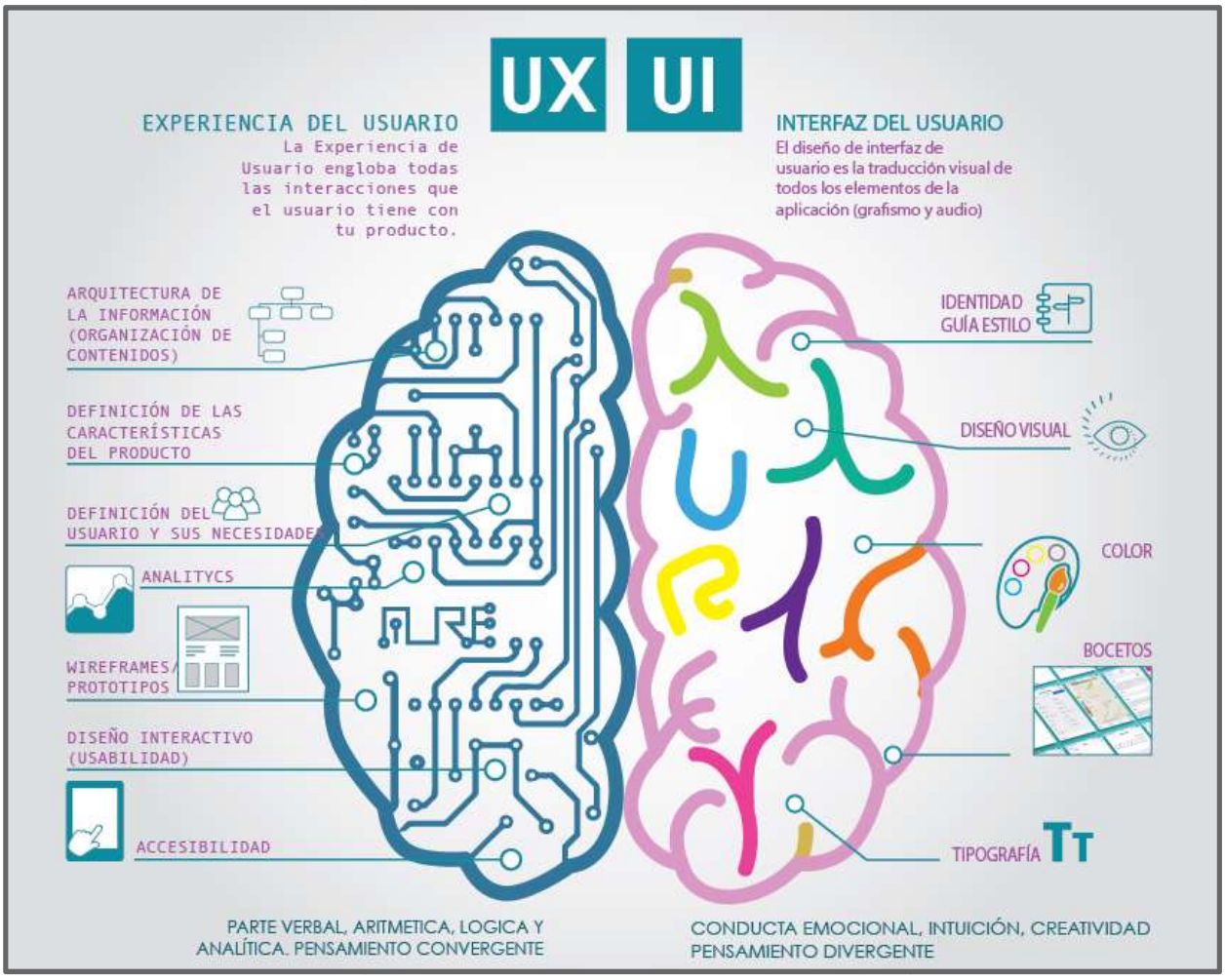

Figura 5. Diagrama UX vs. UI

En este diagrama podemos observar que el perfil de un diseñador UX es mucho más técnico que el de un diseñador UI. UX se encarga de pensar todas las variables y funcionalidades basándose en un estudio previo realizado, en cambio, UI tiene presente toda la parte visual para que el usuario se sienta atraído a la aplicación.

Ironhack Barcelona Workshop

El día 18 de Enero del 2020, hubo un meeting en las instalaciones de Ironhack en Barcelona. Allí se realizó una introducción a UX y un Workshop de Adobe XD.

Se realizó una master class con Laura Feliz, una experta en UX/UI design que trabaja realizando aplicaciones para los servicios de emergencia, como los bomberos.

Web Laura Feliz: https://www.laurafeliz.net/

Su trabajo recalca la importancia de la experiencia del usuario y también lo importante que es tener una idea muy clara al realizar el diseño de una aplicación. Su mayor objetivo es realizar un diseño eficaz y atractivo.

El estudio del mercado y recolectar información básica de los posibles usuarios es algo imprescindible a la hora de crear una aplicación.

El trabajo académico de Laura Feliz, ha sido clave para la realización de este proyecto.

Graus UPC

Graus UPC ha sido diseñada teniendo muy presente la experiencia del usuario y su interfaz. Por eso se ha hecho un pequeño estudio de lo que se necesitaba para hacer el diseño, como puede ser analizar el público que usará la app, o hacer las funcionalidades más accesibles y intuitivas.

- No quedarse con la primera idea que tienes, debes buscarlas todas. Saber quien es tu usuario, el diseñador no es el usuario final, sus ideas son una opinión subjetiva.
- Graus UPC es una app para escoger carrera universitaria, sabiendo esto, su público será joven. La mayoría de personas que entran a la universidad vienen de estudiar un bachillerato, con la edad de 17 y 18 años.

Los estudios técnicos como las ingenierías, el 90% de los estudiantes de primer curso tienen entre 17 y 20 años. (datos procedentes de la web de la UPF, link en la bibliografía) Sabiendo esto se han supuesto estos tres grupos de usuarios:

Grupo 1: entre 17 a 18 años. (mayor uso)
Grupo 2: entre 18 a 25 años. (uso medio)
Grupo 3: entre 25 a 65 años. (uso bajo)

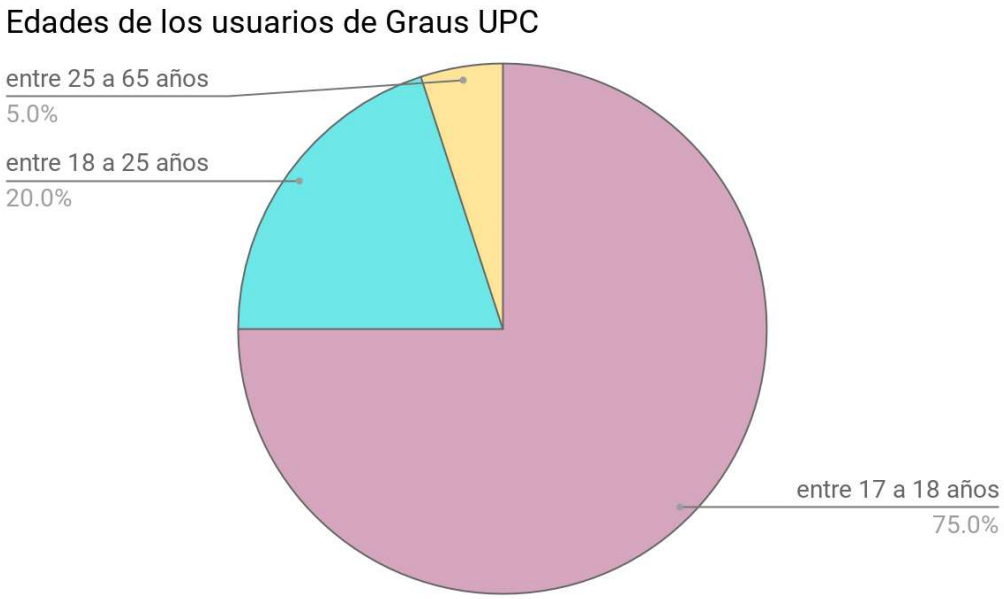

Figura 6. Gráfico

Al analizar quién usará la app con mayor frecuencia se hace un diseño adaptado a sus gustos y con una estética parecida a las apps más usadas por ellos como es Instagram.

- La mayoría de los botones y objetos animados han sido colocados en la parte derecha de la pantalla, para tener un acceso más rápido con el pulgar derecho (el dedo normalmente más usado al navegar con el móvil, sobre todo el público joven).
- Para el diseño de esta app se mostraron bocetos a estudiantes de bachillerato, y se escogieron los elementos que más les gustaban en general.

b. Bocetos

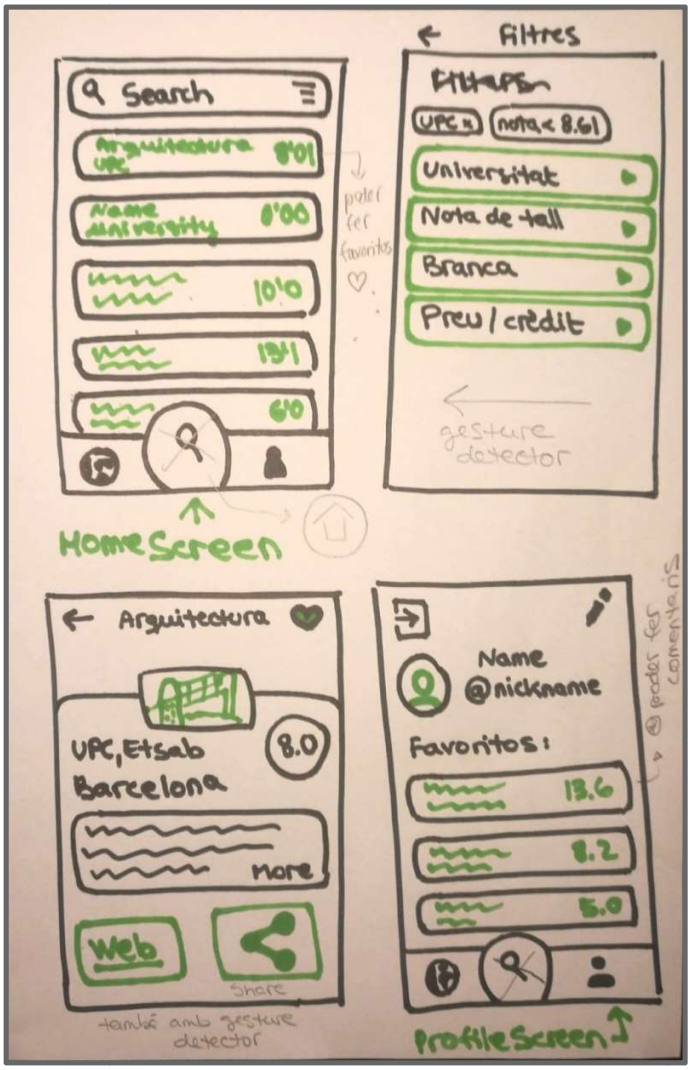

Figura 7. Bocetos

Estas figuras son los bocetos realizados en el workshop de UI/UX. Esta fue la primera idea del diseño de la app, se plantearon las 4 pantallas más principales y fundamentales para el correcto funcionamiento de la app: Pantalla principal con el listado de los grados, la pantalla de filtros, la pantalla de la ficha del grado, y la pantalla del perfil del usuario con el listado de sus grados favoritos.

Luego de estos bocetos se pasó a diseñar las pantallas con el programa Adobe XD. Primero se realizaron pruebas antes de tener los diseños provisionales para empezar a programar la app.

c. Diseño de las pantallas, Adobe XD

Estas son las pantallas que iba a tener la App desde un inicio. Cada pantalla está comentada, explicando sus funcionalidades y valoración general de las mismas.

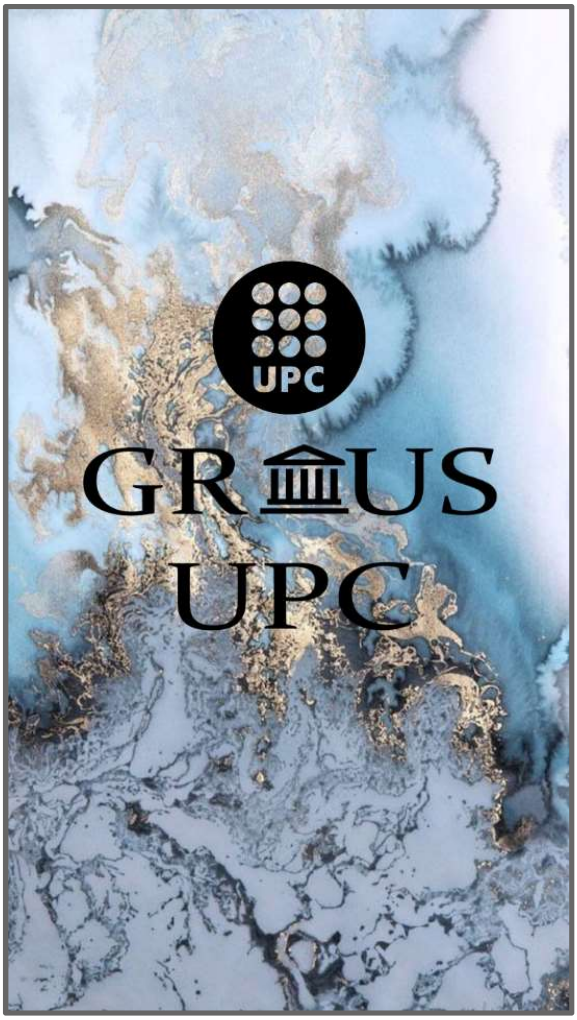

Figura 8
Await Screen or Splash Screen

Esta pantalla tenía que ser la pantalla que se mostraría cada vez que abres la App. Finalmente, se decidió que no era necesario y que es una pérdida de tiempo para el usuario de la App.

Figura 9
Home Screen

Pantalla principal de la app. La función principal es mostrar el listado de los grados de la Universitat Politécnica de Catalunya. Desde esta pantalla se puede acceder a otras dos pantallas desde la barra de debajo (Navbar, Navigation Bar). También se puede hacer una búsqueda de un grado a través de un buscador por palabras y también aplicando filtros. Para aplicar los filtros se debe acceder a otra pantalla. El resultado final no varía demasiado a esta versión, las funcionalidades son las mismas y el diseño es un poco diferente a esta versión.

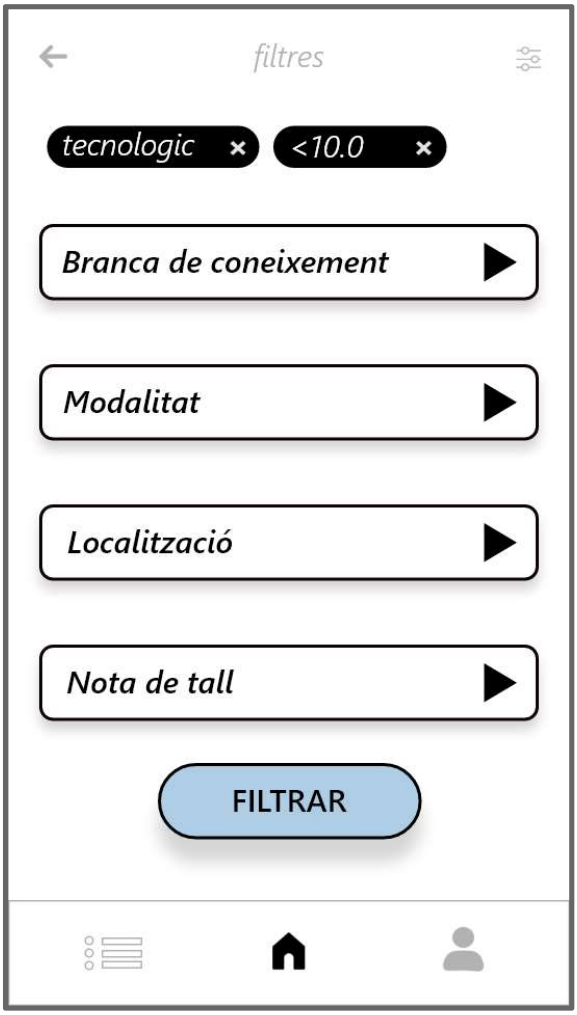

Figura 10
Pantalla Filtros

Pantalla para aplicar los filtros de búsqueda. Se accede desde la Home Screen. El resultado final es casi el mismo, y sus funcionalidades las mismas.

Figura 11
Pantalla Filtros 2

En esta pantalla vemos las opciones para filtrar según cada campo. Se accede a través de la Pantalla de Filtros principal. Para salir de ella se puede usar la flecha que hay arriba (AppBar) o desde la barra de navegación de debajo.

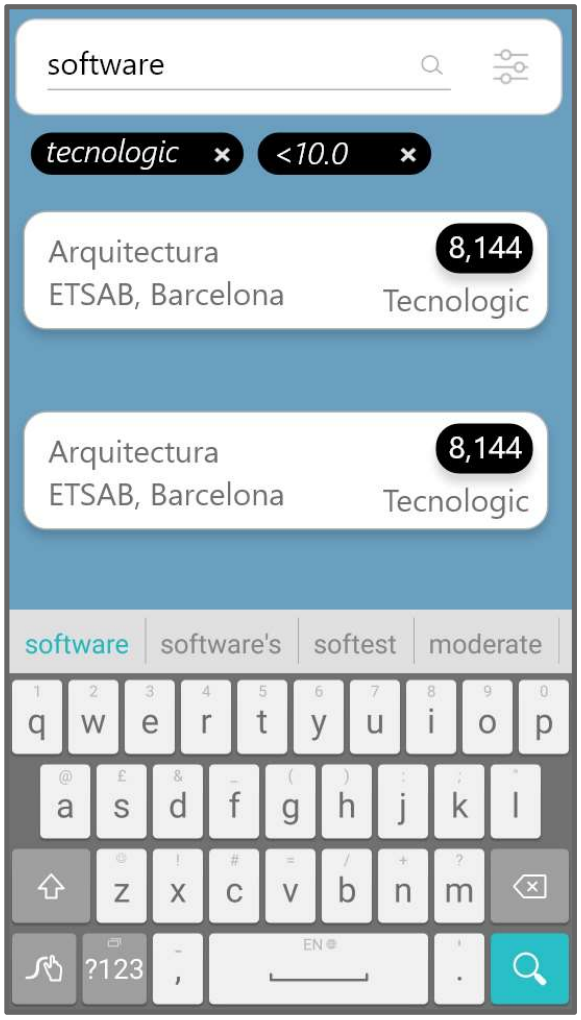

Figura 12
Home Screen (con filtros de búsqueda)

Este es el diseño de cómo quedaría el listado con filtros de búsqueda. Los filtros se pueden eliminar desde esta pantalla.

Figura 13
Pantalla Ficha del grado

Esta es la pantalla donde se muestra la ficha del grado seleccionado desde la pantalla principal con la lista. En esta ficha se muestra el nombre completo del grado, la universidad y localización donde se desarrolla, una pequeña descripción de los estudios, las salidas profesionales de este, la nota de corte, y al final de la ficha un link para ir al sitio web oficial del grado. La barra de arriba permite seleccionar el grado como favorito y se quede guardado en tu cuenta, y también poder compartir el grado desde la aplicación que desees (Gmail, WhatsApp, Telegram, etc). El diseño final de esta pantalla es muy similar a este.

Figura 14
Cuadro de diálogo para añadir el grado a favoritos

Este cuadro de diálogo salta al añadir el grado a favoritos. Desde un inicio se quería poder añadir un comentario a los grados añadidos como favoritos, pero finalmente esta funcionalidad se descartó, podría resultar pesado para los usuarios de la App. Al clicar al icono para añadir a favoritos, si no has iniciado sesión, se te pregunta si deseas iniciar sesión para así poder guardar el grado en tu lista.

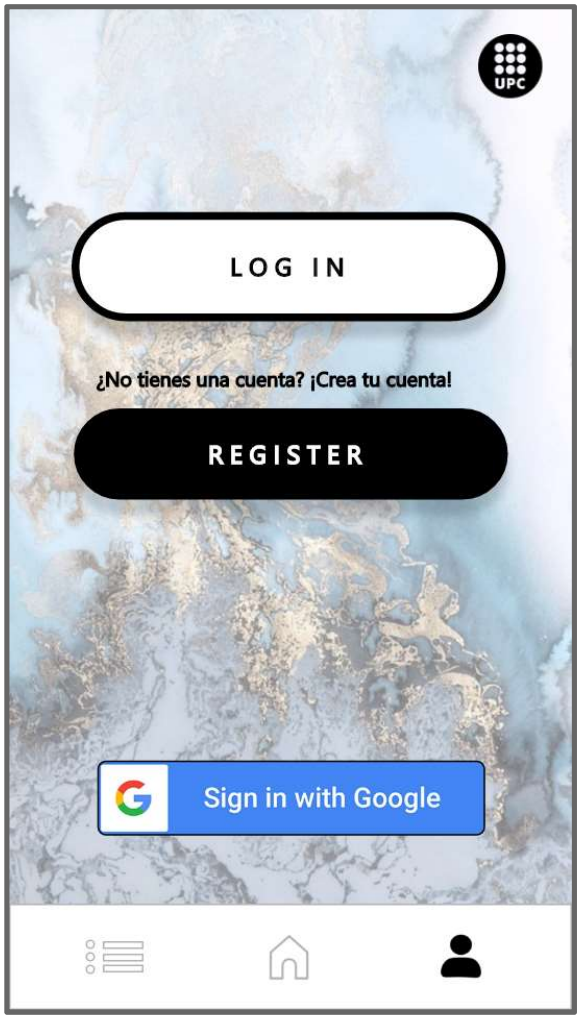

Figura 15
Profile Screen (Usuario no iniciado)

Pantalla del perfil del usuario cuando aún no ha iniciado sesión. Aquí se muestra la opción de iniciar sesión con una cuenta propia de la app y también desde una cuenta de Google.

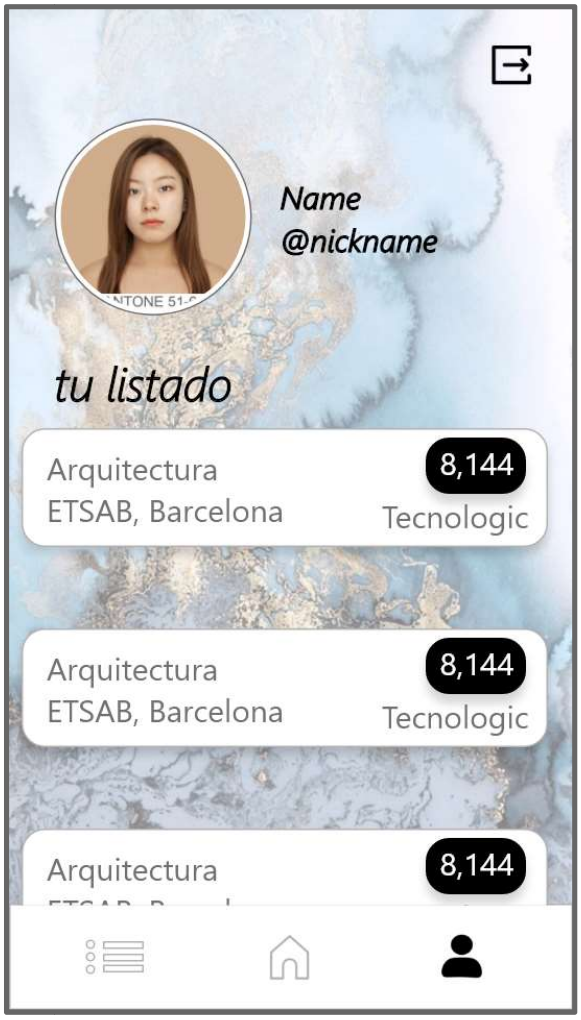

Figura 16
Profile Screen (Usuario ya iniciado)

Pantalla del perfil del usuario una vez ya ha iniciado sesión. Como finalmente el inicio de sesión se realiza únicamente a través de una cuenta Google, la información que tenemos del usuario se obtiene a través de la API de Google. De toda la información finalmente solo se muestra nombre, email y foto de perfil. Esta pantalla también cuenta con un icono para salir de la sesión, y muestra el listado de grados que tenemos guardados como favoritos. El resultado final no difiere mucho a este diseño.

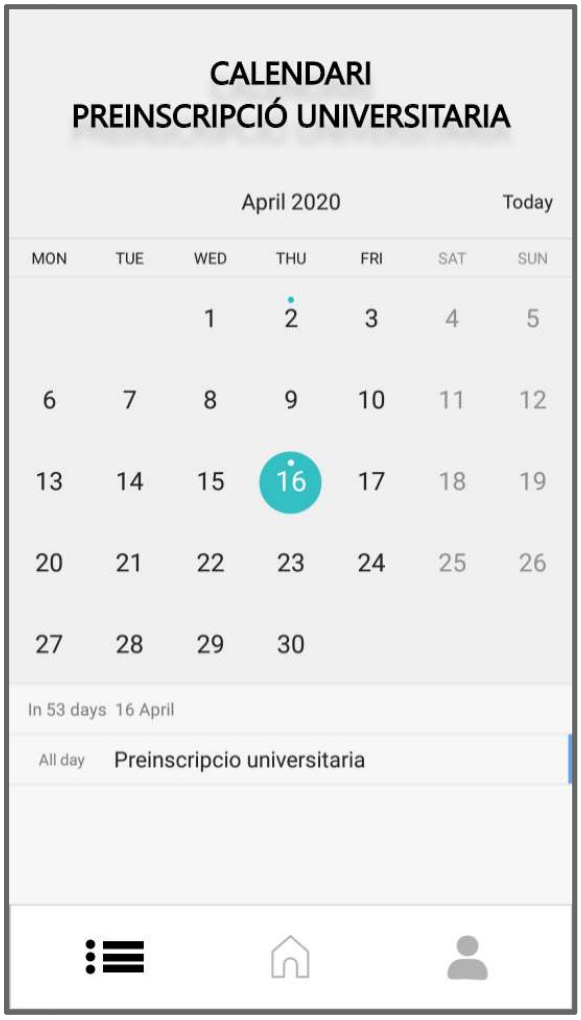

Figura 17
Pantalla con el calendario de las PAU

Pantalla donde se muestra un calendario. Este calendario contiene las fechas que toda persona que quiere acceder a la universidad y que se examine en las PAU necesita saber. La versión final no difiere mucho a esta pantalla.

d. Mapa del resultado final

/* Ver **ANNEX2_MAPA.pdf** */

5. Flutter

a. Introducción a Flutter

Flutter es un SDK* de código fuente abierto de desarrollo de aplicaciones móviles creado por Google. Se utiliza para desarrollar interfaces de usuario para aplicaciones en Android, iOS y Web.

Figura 18. Flutter logo

Flutter en sí fue lanzado en 2017 y la versión 1.0 es de 2018.

Actualmente Flutter está optimizado para aplicaciones móviles 2D que desean ejecutarse tanto Android como iOS. (SDK multiplataforma*)

Desarrollo Rápido

Trae la aplicación a la vida en cuestión de milisegundos con Hot Reload. Usa un completo conjunto de widgets* totalmente personalizables para crear interfaces nativas en cuestión de minutos.

*SDK: Kit de desarrollo de software
SDK es el acrónimo de "Software Development Kit". El SDK reúne un grupo de herramientas que permiten la programación de aplicaciones móviles.
*SDK multiplataforma: Reúne las herramientas que permiten programar aplicaciones móviles de diferentes plataformas.
*widget: pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine. Entre sus objetivos están dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

UI* Expresiva y Flexible

Se crean rápidamente funcionalidades con el foco en la experiencia de usuario nativa. La arquitectura en capas permite una completa personalización, que resultan en un renderizado* increíblemente rápido y diseños expresivos y flexibles.

Rendimiento Nativo

Los widgets de Flutter incorporan todas las diferencias críticas entre plataformas, como el scrolling, navegación, iconos y fuentes para proporcionar un rendimiento totalmente nativo tanto en iOS como en Android.

b. Catálogo Básico de Widgets

Flutter tiene un extenso catálogo de Widgets para realizar una app funcional con un diseño atractivo.

Aquí hay una pequeña muestra de los widgets más usados:

Widgets básicos

Container	Un widget de conveniencia que combina widgets comunes de dibujado, posicionado y dimensionado.
Row	Layout con una lista de widgets hijos en dirección horizontal.
Column	Layout con una lista de widgets hijos en dirección vertical.
Image	Un widget que muestra una imagen.
Text	Una línea de texto con un solo estilo.
Icon	Un ícono de Material Design.
Placeholder	Un widget que dibuja un recuadro que representa dónde se agregaran otros widgets algún día.

*UI: user interface(en), Interfaz del usuario(es)
*renderizado: es un término usado en computación para referirse al proceso de generar una imagen foto realista desde un modelo 3D.
*Widget (en informática) = pequeña aplicación o programa

Estructura de la app y navegación

Scaffold	Implementa la estructura de diseño visual básica de Material Design. Esta clase proporciona API para mostrar drawers, snack bars, y bottom sheets.
AppBar	Una barra de aplicaciones de Material Design. Una barra de aplicaciones consiste en una barra de herramientas y potencialmente otros widgets.
BottomNavigationBar	Las barras de navegación inferiores facilitan explorar y cambiar entre las vistas de nivel superior en un solo toque.
MaterialApp	Un widget de conveniencia que envuelve una cantidad de widgets que comúnmente se requieren para las aplicaciones que implementan Material Design.

Botones

RaisedButton	Un botón elevado de Material Design. Un botón elevado consiste en una pieza rectangular que flota sobre la interfaz
FloatingActionButton	Un botón de acción flotante es un botón de icono circular que flota sobre el contenido para promover una acción principal en la aplicación.
FlatButton	Un botón plano es una sección dibujada en un widget de Componentes de Material que reacciona a los toques rellenándose de color.
IconButton	Un botón de icono es una imagen dibujada en un widget Material que reacciona a los toques rellenando con color.
ButtonBar	Un conjunto de botones dispuestos horizontalmente.

c. Dart

Dart (originalmente llamado Dash) es un lenguaje de programación de código abierto, desarrollado por Google. Fue revelado en la conferencia goto; en Aarhus, Dinamarca el 10 octubre de 2011. El objetivo de Dart no es reemplazar JavaScript como el principal lenguaje de programación web en los navegadores web, sino ofrecer una alternativa más moderna. El espíritu del lenguaje define a Dart como un "lenguaje estructurado pero flexible para programación Web".

Figura 19. Dart logo

El equipo de Flutter evaluó más de una docena de lenguajes de programación y eligió Dart porque coincidía con la forma en que construían las interfaces de usuario.
Aquí hay una lista rápida de las características de Dart que lo hacen indispensable para Flutter:

- Dart es AOT (Ahead Of Time) compilado en un código nativo rápido, predecible, que permite que casi todo Flutter se escriba en Dart. Esto no solo hace que Flutter sea rápido, sino que prácticamente todo (incluidos todos los widgets) se puede personalizar.
- Dart también se puede compilar JIT (Just In Time) para ciclos de desarrollo excepcionalmente rápidos y con un flujo de trabajo que cambia el juego (incluida la popular recarga en caliente popular de Flutter).
- Dart facilita la creación de animaciones y transiciones suaves. Dart puede hacer la asignación de objetos y la recolección de basura sin bloqueos. Y al igual que JavaScript, Dart evita la programación preventiva y la memoria compartida (y, por lo tanto, se bloquea). Debido a que las aplicaciones Flutter se compilan en código nativo, no requieren un puente lento entre reinos (por ejemplo, JavaScript a nativo). También comienzan mucho más rápido.
- Los desarrolladores han descubierto que Dart es particularmente fácil de aprender porque tiene características que son familiares para los usuarios de lenguajes estáticos y dinámicos.

No todas estas características son exclusivas de Dart, pero la combinación de ellas alcanza un punto óptimo que hace que Dart sea especialmente eficiente para implementar Flutter.

6. Firebase

Firebase es la plataforma digital de Google que se utiliza para facilitar el desarrollo de aplicaciones web o móviles de una forma efectiva, rápida y sencilla.

Figura 20. Firebase logo

Su principal objetivo, es mejorar el rendimiento de las apps mediante la implementación de diversas funcionalidades que hacen de la aplicación en cuestión, un producto mucho más manejable, seguro y de fácil acceso para los usuarios.

Todos los datos de Graus UPC están en Firebase, como los usuarios que acceden a ella. La funcionalidad más usada es la de tener una base de datos en tiempo real.

a. Cloud Firestore

Cloud Firestore es una base de datos flexible y escalable para la programación en servidores, dispositivos móviles y la Web desde Firebase y Google Cloud Platform. Al igual que Firebase Realtime Database, mantiene tus datos sincronizados entre apps cliente a través de agentes de escucha en tiempo real y ofrece asistencia sin conexión para dispositivos móviles y la Web, por lo que puedes compilar apps con capacidad de respuesta que funcionan sin importar la latencia de la red ni la conectividad a Internet.

Cloud Firestore también ofrece una integración sin interrupciones con otros productos de Firebase y Google Cloud Platform, incluido Cloud Functions.

¿Como funciona?

Cloud Firestore es una base de datos NoSQL* alojada en la nube a la que pueden acceder tus apps para iOS, Android y Web directamente desde los SDK nativos. Cloud Firestore también está disponible en los SDK nativos de Node.js, Java, Python y Go, junto con las API de REST y RPC.

A partir del modelo de datos NoSQL de Cloud Firestore, almacenan los datos en documentos que contienen campos que se asignan a valores. Estos documentos se almacenan en colecciones, que son contenedores para tus documentos y que puedes usar para organizar tus datos y compilar consultas. Los documentos admiten varios tipos de datos diferentes, desde strings y números simples, hasta objetos anidados complejos.

También puedes crear subcolecciones dentro de documentos y crear estructuras de datos jerárquicas que se ajustan a escala a medida que tu base de datos crece. El modelo de datos de Cloud Firestore admite cualquier estructura de datos que funcione mejor con tu aplicación.
Además, las consultas de Cloud Firestore son expresivas, eficientes y flexibles.

Crea consultas superficiales para recuperar datos en el nivel del documento, sin la necesidad de recuperar la colección completa ni las subcolecciones anidadas. Agrega criterios de orden, filtros y límites a tus consultas o cursores para paginar los resultados. Para mantener actualizados los datos de tus apps sin tener que recuperar toda la base de datos cada vez que haya una actualización, agrega agentes de escucha en tiempo real. Estos te notifican con una instantánea de los datos cada vez que los datos de tus apps cliente escuchan para detectar cambios y recuperan sólo los cambios nuevos.

Se protege el acceso a los datos en Cloud Firestore con Firebase Authentication y las reglas de seguridad de Cloud Firestore para Android, iOS y JavaScript.

*NoSQL: Los sistemas NoSQL se denominan a veces "no solo SQL" para subrayar el hecho de que también pueden soportar lenguajes de consulta de tipo SQL. Es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico de SGBDR (Sistema de Gestión de Bases de Datos Relacionales) en aspectos importantes.

7. GitHub

GitHub es una empresa que ofrece una plataforma de **desarrollo colaborativo de software** para alojar proyectos utilizando el sistema de control de versiones Git.

Figura 21. Git logo

a. Utilidad

GitHub aloja tu repositorio de código y te brinda herramientas muy útiles para el trabajo en equipo, dentro de un proyecto.

Además de eso, puedes contribuir a mejorar el software de los demás. Para poder alcanzar esta meta, GitHub provee de funcionalidades para hacer un fork y solicitar pulls.

Realizar un fork es simplemente clonar un repositorio ajeno (genera una copia en tu cuenta), para eliminar algún error de programación o modificar cosas de él. Una vez realizadas las modificaciones puedes enviar un pull al dueño del proyecto. Éste podrá analizar los cambios que has realizado fácilmente, y si considera interesante tu contribución, adjuntarlo con el repositorio original.

*El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

b. Herramientas

En la actualidad, GitHub es mucho más que un servicio de alojamiento de código. Además de éste, se ofrecen varias herramientas útiles para el trabajo en equipo.
Entre ellas, cabe destacar:

- Una **wiki*** para el mantenimiento de las distintas versiones de las páginas.
- Un **sistema de seguimiento** de problemas que permiten a los miembros de tu equipo detallar un problema con tu software o una sugerencia que deseen hacer.
- Una herramienta de **revisión de código**, donde se pueden añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios realizados en un commit específico.
- Un **visor de ramas** donde se pueden comparar los progresos realizados en las distintas ramas de nuestro repositorio.

*wiki: Sistema de trabajo informático utilizado en los sitios web que permite a los usuarios modificar o crear su contenido de forma rápida y sencilla.

c. GitHub en este proyecto

Para el desarrollo de esta app, todo el código ha sido subido y compartido a través de GitHub, es una forma de que sea dinámico, y de que se puedan hacer modificaciones desde diferentes terminales sin problemas.

El tutor de TFG puede consultar tu código en todo momento y ver tu ritmo de trabajo desde la web.

Si hay algún error en el código puedes retroceder a versiones anteriores sin problemas.

8. Manual general de la app

/* Ver **ANNEX3_MANUAL.pdf** */

9. Programación de la app

Esta aplicación se ha programado en su totalidad en Dart, usando librerías Flutter. Seguidamente se destacan las partes más relevantes de la programación de Graus UPC.

a. Versiones

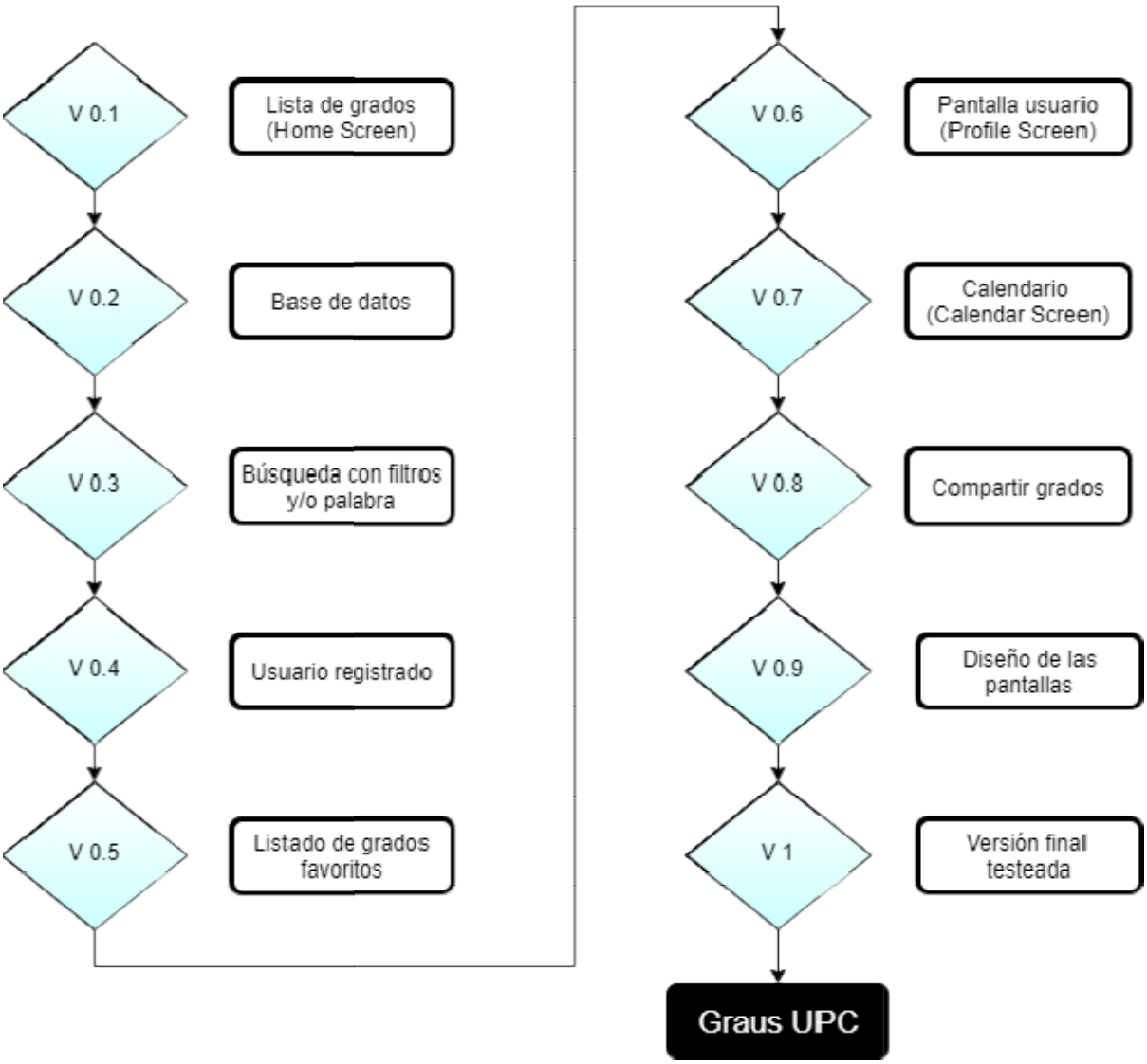

Figura 22. Mapa versiones

v 0.1: Lista de grados (Home Screen)
La primera versión consta sólo del listado de grados. Aparece el nombre del grado, la universidad, el ámbito y la nota de corte.

v 0.2: Base de datos
Añadimos el hecho de leer la información de una base de datos. Además de la lista con los datos también llenamos una descripción más amplia del grado en una ficha, con un enlace a la web oficial y una foto también.

v 0.3: Búsqueda con filtros y/o palabra
Esta versión incorpora un sistema de búsqueda avanzada que permite buscar en la lista. Se pueden elegir varias palabras clave y buscar a la hora diferentes campos para reducir el número de resultados. Un ejemplo sería hacer una búsqueda avanzada de un grado del ámbito tecnológico con una nota inferior a 10.

v 0.4: Usuario registrado
Permite tener tu propio usuario en la app. El inicio de sesión se realiza a través de una cuenta de Google, así todo está vinculado a la base de datos.

v 0.5: Listado de grados favoritos
Añadimos a la versión anterior poder marcar en las fichas de los grados, el grado como favorito. Una vez marcas ese grado como favorito, se te añade a tu lista personal de usuario, como si de una cesta de la compra se tratase.

v 0.6: Pantalla usuario (Profile Screen)
Esta versión permite acceder a la cuenta del usuario utilizando la cuenta de Google. Cada vez cuando se inicia sesión con un usuario se cargan los datos personales de firebase así como los comentarios y grados favoritos.

v 0.7: Calendario (Calendar Screen)
Añadimos una nueva pantalla con la que se visualiza un calendario. Este calendario muestra todas las fechas de la prueba de las PAU, matrículas, prueba, revisiones de examen, etc. Esta pantalla informa al usuario de todos los eventos que no se le pueden pasar a la hora de acceder a la universidad.

v 0.8: Compartir grados
Además de la conectividad entre todas las pantallas para poder navegar por la aplicación incorporará la opción de compartir un grado a través de otras aplicaciones.

v 0.9: Diseño de las pantallas
Realización del diseño de las pantallas de la app. Siguiendo los diseños realizados con Adobe XD, se edita cada pantalla.

v 1: Versión final testeada
La versión final ha sido testeada, y los errores encontrados solucionados.

b. Sincronizar los datos a Firebase

Todos los datos de Graus UPC están en la base de datos de Firebase, para poder acceder a estos servicios lo primero es crear un proyecto accediendo a su página web. En la web de Firebase se debe clicar a "Ir a consola" y a "Crear nuevo proyecto". Después de un proceso de configuración se tiene que navegar al panel de control de nuestro proyecto.

Una vez en el panel, para agregar compatibilidad con Android a nuestro proyecto Flutter se debe registrar la aplicación. Lo más importante en este paso es hacer coincidir el nombre del paquete de Android con el que está dentro de nuestra aplicación. El nombre del paquete android de este proyecto es: `com.example.graus_upc` .

En el registro de la app las claves de firma (SHA1) son necesarias para usar algunas funcionalidades de Google, como permitir que los usuarios de la app puedan hacer Sign In con cuentas de Google. (Más adelante en esta memoria se describe mejor el proceso para poder hacer Sign In con una cuenta de Google.) Para terminar el registró se tiene que introducir el SHA-1 del portátil, que se obtiene consultando en la consola:

```
C:\Users\Usuario\.android>keytool -list -v -alias
androiddebugkey -keystore debug.keystore
```

El siguiente paso es obtener el archivo de configuración de Firebase. Esto es importante ya que contiene las claves API* y otra información crítica para que Firebase la use.

*claves API: es un código que utilizan los programas de ordenador para ponerse en contacto para identificar el programa con el que se comunican, a su desarrollador o al usuario.

Desde Firebase se descarga el archivo `google-services.json` de esta página:

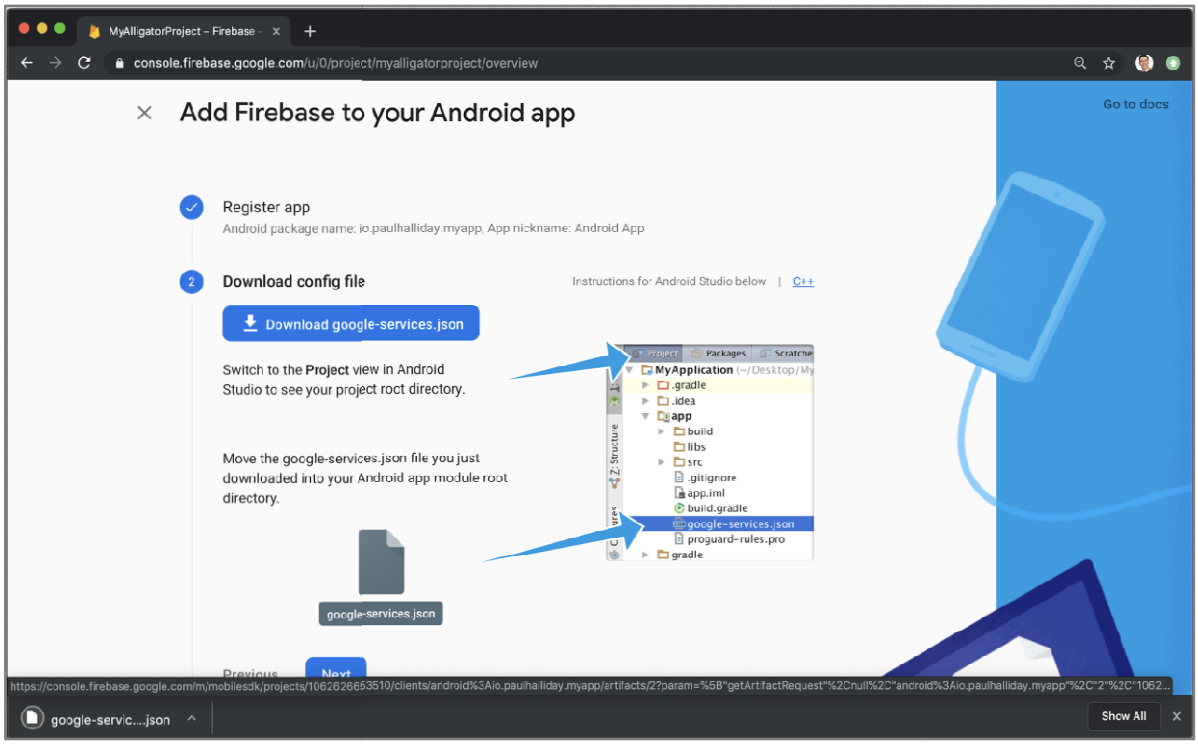

Figura 23. Pantalla Firebase

Una vez descargado el archivo json, se debe guardar en el directorio android / app dentro del proyecto Flutter.

Finalmente se actualiza su configuración de Gradle para incluir el complemento de Servicios de Google. Se encuentra en la carpeta `android/build.gradle` y se modifica para incluir lo siguiente:

```
buildscript {
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }
  dependencies {
    ...
    // Add this line
    classpath 'com.google.gms:google-services:4.3.2'
  }
}
```

```
allprojects {
  ...
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    ...
  }
}
```

Seguidamente también se configura el archivo de nivel de aplicación en `android/app/build.gradle` para incluir:

```
apply plugin: 'com.android.application'

dependencies {
  // add the Firebase SDK for Google Analytics
  implementation 'com.google.firebase:firebase-analytics:17.2.0'
  // add SDKs for any other desired Firebase products
  // https://firebase.google.com/docs/android/setup#available-
libraries
}
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

Con esta actualización, esencialmente se está aplicando el complemento de Servicios de Google, así como observando cómo se pueden activar otros complementos de Flutter Firebase, como Google Analytics.

En este punto se ejecuta la aplicación en un dispositivo o simulador de Android y desde la web de Firebase si todo ha ido bien se muestra que la vinculación funciona correctamente.

c. Layout (Diseño)

La programación en dart tanto diseño como funcionalidades están en el mismo fichero, no como ocurriría en el desarrollo web. Todo se basa en widgets, estos se combinan y forman un árbol, el *Widget Tree*, a través de las propiedades `child` y `children`.

Recordatorio: Un **widget** es una pequeña aplicación o programa. Entre sus objetivos están dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

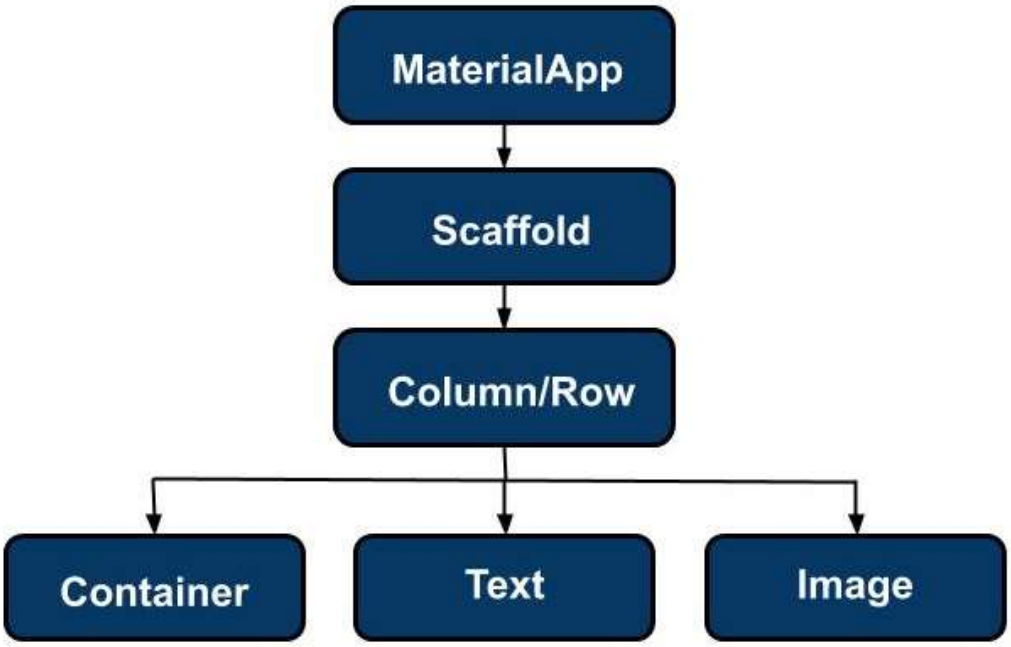

Figura 24. Widget Tree

En un widget la propiedad `child`, introduce un widget al widget principal, y la propiedad `children` un listado de widgets.

```
child: Text('Hola Mundo')
children: <Widget>[Text('Hola'), Text('Mundo')]
```

Los widgets elementales:

- **`MaterialApp:`** Es un widget de conveniencia que envuelve una serie de widgets que comúnmente se requieren para aplicaciones de diseño de materiales.
- **`Scaffold:`** Es un widget que proporciona un marco para implementar el material básico del diseño de la aplicación.
- **`Column:`** Es un widget que muestra a sus hijos (`children`) en una matriz vertical.

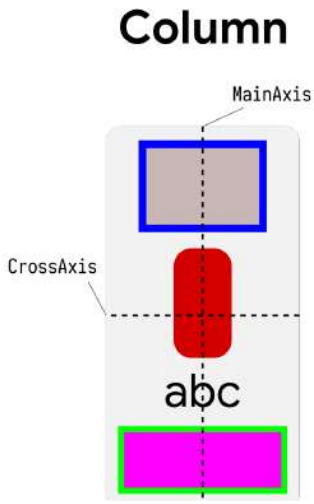

Figura 25. Column
MainAxis: eje vertical
CrossAxis: eje horizontal

- **`Row:`** Es un widget que muestra a sus hijos (`children`) en una matriz horizontal.

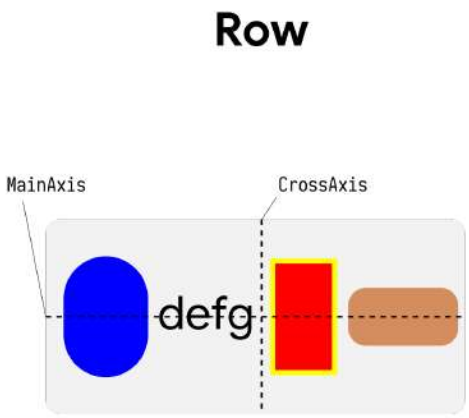

Figura 26. Row
MainAxis: eje horizontal
CrossAxis: eje vertical

d. Listado de grados

La principal funcionalidad de la app es mostrar el listado completo de la oferta de grados de la UPC. Para mostrar este listado tanto en la pantalla principal, como el listado de favoritos en la pantalla de usuario se han usado funciones parecidas. Las partes del código más relevantes son:

```
return ListView.builder(
            itemExtent: 100,
            itemCount: grausFiltrats.length,
            itemBuilder: (context, index) =>
                _grau(context, grausFiltrats[index]),
          );
```

Con este `return`, se devuelve un constructor de listas. Este constructor, necesita un objeto con el que crear la lista, en este caso se le pasa un widget(_grau) con grausFiltrats, una lista con los grados ya filtrados.

```
Widget _grau(BuildContext context, Grau grau) {
  return Card(
    color: Colors.white70.withOpacity(0.7),
    child: Padding(
      padding: const EdgeInsets.all(12.0),
      child: InkWell(
        child: Column( // Column
        onTap: () {
          Navigator.of(context).push(
            MaterialPageRoute(
              builder: (context) {
                return FichaScreen(grau.nom, grau.ambit, grau.loc, grau.link,
                    grau.nota.toString(), grau.objectius, grau.foto, grau.id);
              },
            ), // MaterialPageRoute
          );
        },
      ), // InkWell
    ), // Padding
  ); // Card
 }
}
```

Figura 27. Screenshot del código

Este es el widget **`_grau`**, aquí se define el aspecto de la lista.

Para la visualización de la lista se usan 4 widgets clave:

- **`Card`:** Produce una tarjeta.
- **`Padding`:** Produce el espacio entre el contenido del elemento y su borde.
- **`InkWell`:** Añade la propiedad de que cada tarjeta de cada grado sea un botón, y puedas acceder a otra página al clicar.
  Esta función se da en la línea que pone **`onTap: () {`** , con el `Navigator` podemos navegar a otra página de la aplicación.
  En este caso saltamos a la página con la ficha de un grado en concreto, la información del grado en concreto se pasa en el `return` marcado en azul.
- **`Column`:** Dentro de este widget se define el texto y diseño de las tarjetas. Los objetos los ordena verticalmente como si de una columna se tratase. En la captura de pantalla el widget `Column` está minimizado.

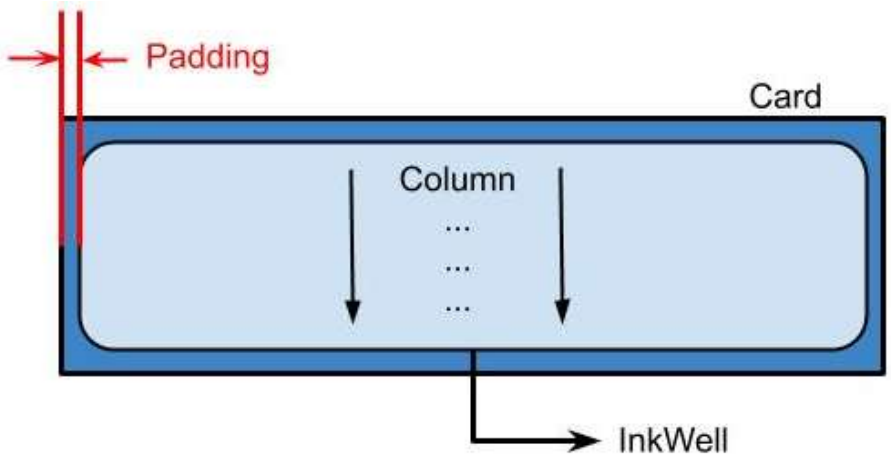

Figura 28. Esquema de los widgets del listado

e. Sign In with Google

El inicio de sesión con una cuenta Google no fue una funcionalidad fácil de añadir. Primero se debe ir al apartado de `authentification` del proyecto Firebase, y desde allí añadir el permiso para hacer el inicio de sesión con Google.

Aquí tenemos la parte del código más importante en el proceso de inicio de sesión con Google:

```
final FirebaseAuth _auth = FirebaseAuth.instance;
final GoogleSignIn googleSignIn = GoogleSignIn();

final databaseReference = Firestore.instance;
```

Con estas tres variables de tipo final obtendremos la autentificación de Firebase, la función para iniciar sesión con Google y la referencia a la base de datos de Firebase.

```
void createRecord(String uid) async {
  try {
    await databaseReference
        .collection("Users")
        .document(uid)
        .updateData({'preferits': FieldValue.arrayUnion([])});
  } on PlatformException catch (e) {
    await databaseReference
        .collection("Users")
        .document(uid)
        .setData({'preferits': FieldValue.arrayUnion([])});
  }
}
```

Esta función llamada `createRecord()` es usada para tener registro de nuestros usuarios en la base de datos de Firebase, ya que sin esta función el usuario no queda en la base de datos, tan solo queda registrado en el apartado de autentificación del proyecto de Firebase.
Al registrar el usuario, también creamos el array* que tendrá los ids(códigos de identificación) de los grados añadidos a favoritos.

*array: Un conjunto indexado de elementos relacionados.

```
Future<User> signInWithGoogle() async {
  GoogleSignInAccount googleSignInAccount;
  try {
    googleSignInAccount = await googleSignIn.signIn();
  } on PlatformException catch (e) {
    print("PlatformException Error: ${e.toString()}");
    return null;
  }
  final GoogleSignInAuthentication googleSignInAuthentication =
      await googleSignInAccount.authentication;
  final AuthCredential credential = GoogleAuthProvider.getCreden
tial(
    accessToken: googleSignInAuthentication.accessToken,
    idToken: googleSignInAuthentication.idToken,
  );
  final AuthResult authResult = await _auth.signInWithCredential
(credential);
  final FirebaseUser user = authResult.user;

  assert(user.email != null);
  assert(user.displayName != null);
  final myuser = User(
    user.uid,
    user.displayName,
    user.email,
    user.photoUrl,
  );
  assert(!user.isAnonymous);
  assert(await user.getIdToken() != null);

  final FirebaseUser currentUser = await _auth.currentUser();
  assert(user.uid == currentUser.uid);

  createRecord(myuser.uid);
  return myuser;
}
```

Este es el código que permite el inicio de sesión con Google.

Al principio hay un `try{ }` on `catch (e) { }` porque la línea dentro del try es problemática y puede dar errores, con el catch podemos capturar el error (`e`). Una vez superada esta línea solo queda autentificar el usuario en Firebase, y rellenar los datos dentro de nuestra variable **`myuser`** tipo usuario.

Una vez nuestro usuario ha estado autentificado y creado, podemos llamar a la función **`createRecord ()`** para tener registro de nuestro usuario en la base de datos.

```
void signOutGoogle() async {
  await googleSignIn.signOut();

  print("User Sign Out");
}
```

Finalmente, comentar esta pequeña función que permite la desconexión del usuario dentro de la app.

f. Providers

Provider, es un paquete el cual tiene como función, permitirnos manejar el estado de nuestra app. En Flutter para poder construir una App, y alterar el aspecto visual de la misma, debemos trabajar con el "Estado de la App". La forma más sencilla de hacer esto, es mediante el uso de "`Stateful Widgets`" (con estado) y la función de "`setState`" (establecer estado).

Esto puede funcionar para alguna aplicación muy pequeña, sin embargo cuando estamos construyendo una app que contiene varias pantallas y hay que acceder a esas propiedades en todas estas pantallas, el proceso se vuelve muy complejo. Es por esto que los providers han sido clave en este proyecto.

Visualmente el funcionamiento de un provider es este:

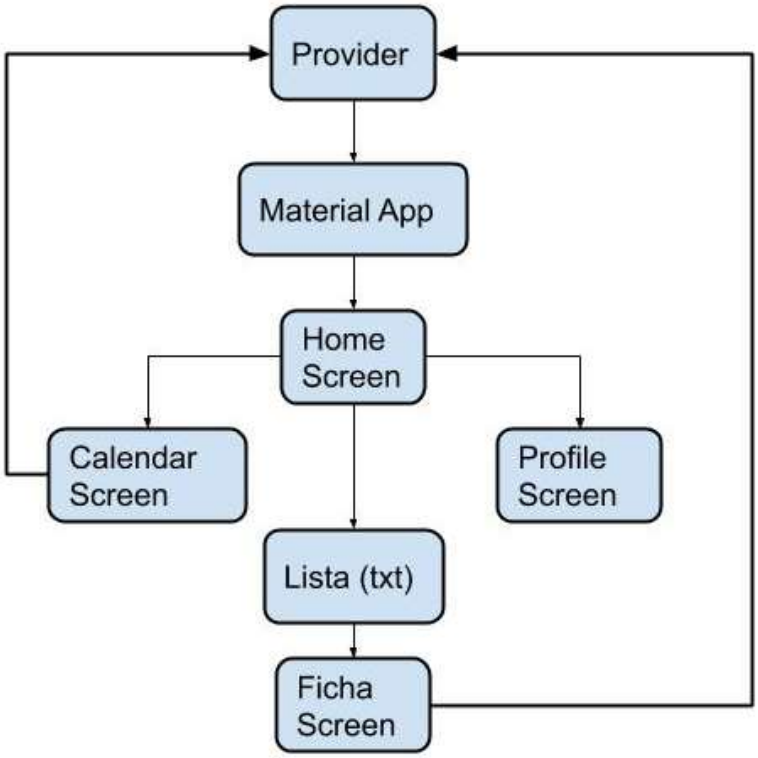

Figura 29. Funcionamiento Provider

Se puede acceder a los objetos del provider y modificar su estado desde cualquier pantalla. Esto resulta muy útil cuando necesitas el estado de un objeto en diferentes pantallas.

En el archivo `Main` (archivo principal), antes de llamar a la MaterialApp, usamos 3 providers. Este es el código:

```
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider<Filtrar>(
      create: (context) => Filtrar(),
      child: ChangeNotifierProvider<LocsList>(
        create: (context) => LocsList(),
        child: ChangeNotifierProvider<UserAuthState>(
          create: (context) => UserAuthState(),
          child: MaterialApp(
            debugShowCheckedModeBanner: false,
            home: Scaffold(body: HomeScreen()),
          ),
        ),
      ),
    );
  }
}
```

Los providers son:

- `Filtrar()`
  Se usa para acceder a la clase con los campos del filtro de la lista.
- `LocsList()`
  Se usa para acceder a la clase con el listado de las localizaciones de las universidades.
- `UserAuthState()`
  Se usa para acceder a la clase con los campos de autentificación del usuario. Hay muchas pantallas donde necesitas saber la identificación del usuario o saber si está registrado en la app.

Aquí tenemos en más detalle el `UserAuthState()`:

```
class UserAuthState with ChangeNotifier {
  User user;
  get check => user != null;

  void signIn([Function callback]) {
    signInWithGoogle().then((User user) {
      this.user = user;
      notifyListeners();
      if (callback != null) {
        callback();
      }
    });
  }

  void setUser(User user) {
    this.user = user;
    notifyListeners();
  }

  void signOut() {
    signOutGoogle();
    user = null;
    notifyListeners();
  }
}
```

Podemos observar que es una clase, con dos objetos: un usuario tipo usuario y un booleano* llamado `check` para saber si tenemos usuario, si tenemos usuario `check` será true y si no tenemos usuario será false.

También tiene tres funciones que participan en la autentificación del usuario, su inicio de sesión y su desconexión. Estas funciones usan otras funciones comentadas en el apartado anterior (Sign In with Google).

*booleano: El tipo de datos booleanos es un tipo de datos que tiene uno de los dos valores posibles, generalmente denotados verdadero y falso.

g. Stream Builder

Un `StreamBuilder` es un widget que se ha usado muchas veces a lo largo del proyecto. Es un widget que se basa en la última instantánea de interacción con un `Stream`.
Un `Stream` es un flujo que proporciona una forma de recibir una secuencia de eventos. Cada evento es un evento de datos, también llamado un elemento de la secuencia, o un evento de error, que es una notificación de que algo ha fallado. Cuando un flujo ha emitido todo su evento, un solo evento "hecho" notificará al oyente que se ha alcanzado el final.

Resumiendo, este widget es un medio para responder a una procesión asíncrona de datos. Un `StreamBuilder` es un `Stateful Widget` (con estado) y, por lo tanto, puede mantener un ***resumen en ejecución*** y/o registrar y anotar el ***último elemento de datos*** de una secuencia de datos.

Este widget lo hemos usado sobretodo para la carga de datos de Firebase. Se han usado para:

- Home Screen → Cargar los **datos de la lista** de grados completa.
- Profile Screen → Cargar la **lista de ids de grados favoritos** de cada usuario.
- Profile Screen → Cargar la **lista de grados completa** y filtrar esta con la del usuario.
- Ficha Screen → Cargar la **lista de ids de grados favoritos** de cada usuario

Aquí podemos ver el `StreamBuilder` para crear la lista principal:

```
body: StreamBuilder(
      stream:
Firestore.instance.collection('Graus').orderBy('nom').snapshots(),
      builder: (context, snapshot) {
       if (!snapshot.hasData) return const Text('Loading...');

       List<DocumentSnapshot> documents = snapshot.data.documents;
       List<Grau> graus =
           documents.map((doc) => Grau.fromFirestore(doc)).toList();
```

Este `StreamBuilder` se usa para cargar el listado de los grados con la información de cada grado. En el `stream` se llama a la colección *Graus* ordenando este objeto por orden alfabético según el campo *nom*.

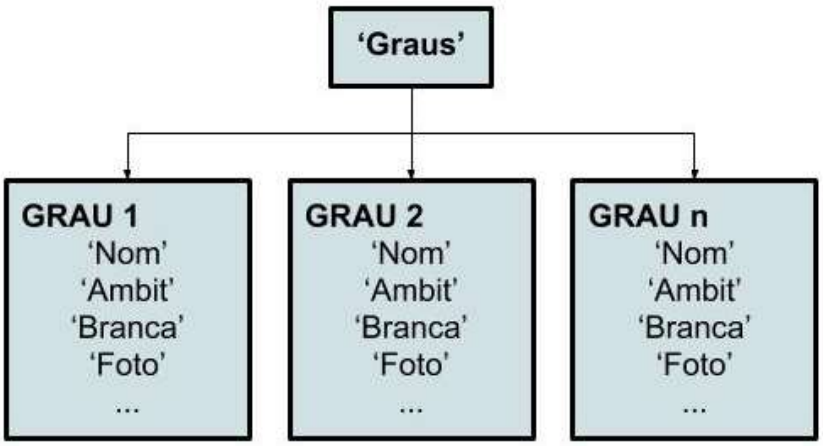

Figura 30. Colección 'Graus'

Todos estos datos obtenidos con el `stream` se guardan en un snapshot (instantánea) como si de fotografías se tratase. Primero se obtienen los documentos de la snapshot y se guardan en la variable **documents**, que es un mapa, luego este mapa se pasa a listado. Este listado, ya es el listado completo de los grados de la UPC de la base de datos.

h. Listado de favoritos

Una de las funcionalidades de Graus UPC es poder guardar en un listado privado los grados que más te gustan. Esta lista queda guardada en nuestra base de datos, y así puede ser mostrada en la pantalla del usuario. Los datos de esta lista se modifican en tiempo real, pudiendo mostrar el listado correcto aunque haya habido modificaciones hace segundos.

La programación de esta funcionalidad se ha basado en `StreamBuilders`. Seguidamente, se muestra un `StreamBuilder` del archivo de la pantalla de la ficha del grado:

```
return StreamBuilder(
        stream: Firestore.instance
            .collection('Users')
            .document((authState.check ? authState.user.uid : 'joker
'))
            .snapshots(),
        builder: (context, snapshot) {
          if (!snapshot.hasData) return const Text('Loading...');

          Map idsGraus = snapshot.data.data;

          List<dynamic> listIds = [];

          idsGraus.forEach((k, v) => listIds.add(v));

          bool favourite = false;

          for (int i = 0; i < listIds[0].length; i++) {
            if (id == listIds[0][i]) {
              favourite = true;
            }
          }
```

Este `StreamBuilder` de la página de la ficha se usa para cargar el listado de los ids de los grados favoritos de un usuario en concreto. Cuando nos referimos al documento de la colección, es el id del usuario, seguidamente se muestra un diagrama de esta colección:

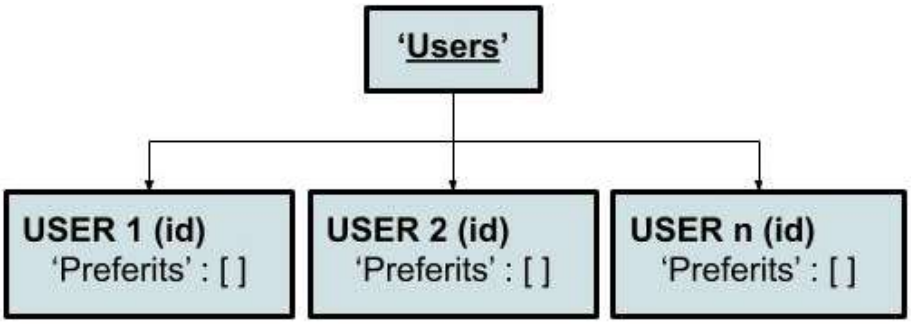

Figura 31. Colección 'Users'

La colección está formada por un documento por cada usuario, dentro de este documento hay un array con el listado de ids de los grados favoritos (`preferits [ ]`). En el `stream` primero debemos comprobar si hay un usuario, si no hay un usuario, se debe usar un usuario comodín llamado `'joker'`, esto es debido a que un `stream` nunca puede no devolver nada, este joker devuelve un listado vacío, pero devuelve un listado.

En el apartado de `builder` se trata el listado de favoritos, ya que llega como un mapa. Una vez ya es una lista, se debe comprobar si el grado de la ficha que se ha abierto esta en favoritos, y así saber si se debe activar el corazón de la pantalla de la ficha. Esta comprobación se realiza con el `for` final.

```
void addPreferitsArray(uid, grauid, favIcon) async {
      DocumentReference docRef =
          Firestore.instance.collection('Users').document(uid);
      if (favIcon == false) {
        docRef.updateData({
          'preferits': FieldValue.arrayUnion([grauid])
        });
      } else {
        try {
          docRef.updateData({
            'preferits': FieldValue.arrayRemove([grauid])
          });
        } catch (e) {
          print("PlatformException Error: ${e.toString()}");
        }
      }
    }
```

Esta función es llamada cuando se clica el **botón de favoritos** y el usuario ya ha iniciado sesión en la app. Su objetivo es modificar el listado de favoritos del usuario, en el primer `if` añadiendo el id del grado a la lista (activación del corazón) y el `else` eliminando el id del grado de la lista (desactivación del corazón).

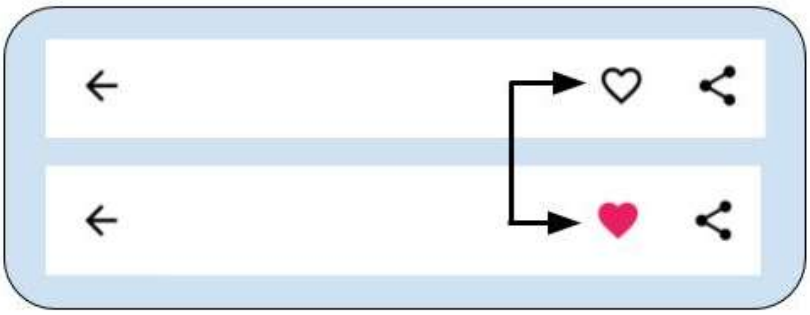

Figura 32. Visualización icono favoritos

i. Calendario

En una de las pantallas principales tenemos un calendario. En este calendario se muestran las fechas para todos los eventos relacionados con las PAU (pruebas de acceso a la universidad).

Las variables principales en el calendario son de tipo `Time` (tiempo). Aquí tenemos un ejemplo:

```
final _selectedDay = DateTime.now();
```

Esta variable es la fecha del día en que abrimos la app, así siempre se muestra en qué día estamos.

Para la programación de esta pantalla se ha usado un paquete de Flutter llamado `table_calendar` (*ver bibliografía*). Este calendario Flutter es altamente personalizable y repleto de funciones con gestos, animaciones y múltiples formatos. Para poder usarlo se deben instalar dependencias al proyecto e importar el paquete en el archivo donde será usado.

```
import 'package:table_calendar/table_calendar.dart';
```

Con este paquete se ha escrito este widget:

```
Widget _buildTableCalendar() {
    return TableCalendar(
      calendarController: _calendarController,
      events: _events,
      holidays: _holidays,
      startingDayOfWeek: StartingDayOfWeek.monday,
      calendarStyle: CalendarStyle(
        selectedColor: Colors.pink,
        todayColor: Colors.pink[100],
        markersColor: Colors.brown[700],
        outsideDaysVisible: false,
      ), headerStyle: HeaderStyle(
        formatButtonTextStyle:
      TextStyle().copyWith(color: Colors.white, fontSize: 15.0),
        formatButtonDecoration: BoxDecoration(
          color: Colors.blueGrey,
          borderRadius: BorderRadius.circular(16.0),
        ),
```

```
      ),
      onDaySelected: _onDaySelected,
      onVisibleDaysChanged: _onVisibleDaysChanged,
      onCalendarCreated: _onCalendarCreated,
    );
  }
```

El elemento ***_calendarController*** se obtiene con una función del paquete de flutter, ***_events*** es un mapa donde hay las fechas de las PAU, y ***_holidays*** también es un mapa con los días festivos.

Todas las otras variables conforman el calendario en sí y su diseño.

10. Conclusiones

Este proyecto ha sido una gran oportunidad para aprender muchísimo sobre como programar con Flutter y también cómo trabajar con la base de datos de Firebase.

El código se ha intentado hacer lo más breve y comprensible posible. Con esta forma de trabajar te das cuenta de la importancia de hacer funciones objetivas, con nombres muy aclaratorios, para que así el código sea fácil de entender por otras personas. La eficiencia del código es vital cuando un proyecto empieza a crecer, y el orden a la hora de redactar es tu mejor aliado.

Al ser un proyecto realizado por dos personas es muy importante la comunicación y la constancia. Los softwares y aplicaciones normalmente están realizados por equipos de programadores, por este motivo he encontrado muy interesante tener la oportunidad de ver de primera mano cómo hacer código simultáneamente con otra persona. Nos hemos entendido a la perfección durante todo el proyecto, hemos estado en todo momento en contacto y nos hemos repartido las tareas equitativamente.

También ha sido un gran reto, algunos días ha sido difícil ponerse a trabajar debido a tener que trabajar desde casa, y han habido partes de la programación arduas de realizar.

La versión final de esta aplicación podría mejorarse mucho más con más tiempo y recursos. Por este motivo, veo este trabajo como un proyecto abierto a mejoras, y me gustaría pulir las funcionalidades, añadir nuevas, y incluso crear un sitio web para esta aplicación.

Cada semana nos hemos reunido telemáticamente con nuestro tutor para mostrarle los avances y preguntar dudas. En general, todo el trabajo ha ido realmente bien, y estoy muy contenta con la experiencia. A pesar del estado de alarma, y de no poder vernos físicamente, todo ha ido muy bien.

11. Bibliografía

3D CAD Portal. ”RENDER-RENDERING-RENDERIZADO” [en línia]. Fuente: Wikipedia. © Copyright “2020” 3DCadPortal - 1er portal de información CAD CAM CAE en español. [Fecha consulta: 28/05/2020]. Disponible en: <http://www.3dcadportal.com/rendering.html>.

Flutter by Google. ”Flutter” [en línia]. Autor: Google. Creative Commons Attribution 4.0 International License, BSD license. [Fecha consulta: Del 24/02/2020 al 29/06/2020]. Disponible en: <https://flutter.dev/>.

Flutter by Google. “Flutter SDK” [en línia]. Autor: Google. Flutter 1.17.4. 2020-06-18, 15:47. Creative Commons Attribution 4.0 International License, BSD license. [Fecha consulta: Del 27/03/2020 al 29/06/2020]. Disponible en: <https://api.flutter.dev/>.

Firebase by Google. “Firebase” [en línia]. Autor: Google. 2020-02-18. Creative Commons Attribution 4.0 International License, BSD license. [Fecha consulta: Del 1/03/2020 al 29/06/2020]. Disponible en: <https://firebase.google.com/>.

GitHub. “GitHub” [en línia]. Autor: GitHub. © 2020 GitHub, Inc. [Fecha consulta: Del 16/02/2020 al 29/06/2020]. Disponible en: <https://github.com/>.

StackOverflow. “StackOverflow” [en línia]. Autor: Stack Overflow Company, since 2008. *StackOverflow* [en línia]. site design / logo © 2020 Stack Exchange Inc; user contributions licensed under cc by-sa. rev 2020.6.26.37146. [Fecha consulta: Del 5/03/2020 al 2/06/2020]. Disponible en: <https://stackoverflow.com/>.

Pub.dev. “*table_calendar package”* [en línia]. Table_calendar, 14/03/2020. Google Terms of Service. [Fecha consulta: 25/04/2020]. Disponible en: <https://pub.dev/packages/table_calendar>.

Paul Halliday. ”How to set up Firebase with Flutter for IOS and Android Apps” [en línia]. Community. Posted September 27, 2019. © 2020 DigitalOcean, LLC. All rights reserved. [Fecha consulta: 16/05/2020]. Disponible en: <https://www.digitalocean.com/community/tutorials/flutter-firebase-setup>.

Aureliano Moret. "Ux frente Ui" [en línia]. 1.0. aureliano.com, 12/04/2020. Copyright © Aureliano Moret. [Fecha consulta: 25/02/2020]. Disponible en: <https://aureliano.com/ux-frente-a-ui/>.

Natalia Di Gifico. "¿POR QUÉ GMAIL LIDERA EL RANKING DE PLATAFORMAS DE EMAIL?" [en línia]. 1.0. Meriti, 8/04/2019. © 2018, MERITI Casa Central. [Fecha consulta: 26/05/2020]. Disponible en: <https://nube.meriti.com/blog/gmail-primero-entre-los-webmails>.

UPF. "Estudiants per edat i estudis (en percentatges)" [en línia]. 1.0. UPF. © Universitat Pompeu Fabra Barcelona. [Fecha consulta: 26/02/2020]. Disponible en: <https://www.upf.edu/web/universitat/estudiants-per-edat-i-estudis-en-percentatges->.

Educaweb. "Descobreix què vols estudiar" [en línia]. Educaweb.cat. © Copyright Educaonline S.L. 1998-2020. [Fecha consulta: 15/02/2020]. Disponible en: <https://www.educaweb.cat/>.

Generalitat de Catalunya. "*Canal Universitat*" [en línia]. Generalitat de Catalunya, 28/06/2018. ©Generalitat de Catalunya. [Fecha consulta: 15/02/2020]. Disponible en: <http://universitats.gencat.cat/ca/inici>.

Colwith Road, S. L. *"unportal"* [en línia]. © COLWITH ROAD, S. L. Todos los derechos reservados, 2010. [Fecha consulta: 15/02/2020 Disponible en: <https://graus.unportal.net/wb/unportal/es/buscador/index.html >.

Infoeducación. "Carreras Universitarias en España 2020/2021" [en línia]. © Copyright 2020, All Rights Reserved - Aviso legal y privacidad. [Fecha consulta: 15/02/2020]. Disponible en: <https://infoeducacion.es/carreras-universitarias-espana/>.

COMPÁS MEDITERRÁNEO, S.L. "Carreras Universitarias en España | Y Ahora Qué" [en línia]. © 2003-2020 Compás Mediterráneo S.L. - Diego de León 47 - 28006 Madrid [ESPAÑA]. [Fecha consulta: 15/02/2020]. Disponible en: <https://yaq.es/carreras-universitarias>.

COMPÁS MEDITERRÁNEO, S.L. *"Notas de Corte 2020"* [en línia]. © 2003-2020 Compás Mediterráneo S.L. - Diego de León 47 - 28006 Madrid [ESPAÑA]. [Fecha consulta: 15/02/2020]. Disponible en: <https://notasdecorte.es/>.


UPC